A FRAMEWORK for MULTI-ROBOT COORDINATION

A DISSERTATION

Submitted in partial fulfilment of the

requirements for the award of the degree

of

INTEGRATED DUAL DEGREE

in

COMPUTER SCIENCE AND ENGINEERING

BY

ARUN A R

Department of Computer Science and Engineering

Indian Institute of Technology Roorkee

Roorkee- 247667, India

MAY, 2018

A FRAMEWORK
for
MULTI-ROBOT COORDINATION

A DISSERTATION

Submitted in partial fulfilment of the

requirement for the award of the degree of

INTEGRATED DUAL DEGREE

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

ARUN A R

Enrolment No: 11211004

under the supervision of

Dr. RAJDEEP NIYOGI

(Associate Professor)

Department of Computer Science and Engineering

Indian Institute of Technology Roorkee

Roorkee- 247667, India

MAY, 2018

AUTHOR'S DECLARATION

I declare that the work presented in this dissertation with title, **A framework for Multi-Robot Coordination** towards the fulfillment of the requirements for award of the degree of **Integrated Dual Degree in Computer Science & Engineering**, submitted to the Department of Computer Science & Engineering, **Indian Institute of Technology Roorkee, India,** is an authentic record of my own work carried out during the period from June 2017 to May 2018 under the guidance of **Dr.Rajdeep Niyogi, Associate Professor,** Department of Computer Science and Engineering, Indian Institute of Technology, Roorkee.

Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Date :

Place : IIT Roorkee

Arun A R (11211004)

IDD, C.S.E,

Indian Institute of Technology, Roorkee.

CERTIFICATE

This is to certify that the statement made by the candidate in the declaration is correct to the best of my knowledge and belief.

Date :

Place : Roorkee

Dr. Rajdeep Niyogi
(Supervisor)
Associate Professor
Department of Computer Science & Engineering
Indian Institute of Technology Roorkee

ACKNOWLEDGMENT

First of all, I bow my head in humility to the Almighty, most graceful and loving, the creator of the universe, for providing me with this opportunity to work with the intelligentsia and enabling me to reach far beyond my own, restricted ambit of thought and action it is my great pleasure to express my sincere and profound thanks to my learned supervisor Dr. Rajdeep Niyogi, Associate professor - Department of Computer Science and Engineering, IIT Roorkee, who has given me enormous support, guidance and encouragement. He has provided me many useful suggestion and guidance during the study. His observations and comments helped me to establish the overall direction to the research and to move forward with investigation in depth. He has helped me greatly and been a source of knowledge.

I am very much indebted to my supervisor, for his continuous encouragement and support. He is always ready to help with a smile. I am also thankful to all the professors at the department for all their patience and support during difficult times in my studies.

Secondly, I would like to thank my father Ravanan A and my mother Manimehalai M for their unwavering support and help throughout the years. I would specially like to thank my friend Mr. Amar Nath Dhebla whose technical expertise and moral support helped me to push my own goals further. I would like to thank administrative, technical and non-technical staff members of the Department who have been kind enough to help in their respective roles.

AR Arun
Er. No. 11211004

ABBREVIATIONS

ARGoS	Autonomous Robots Go Swarming
MAP	Multi Agent Planning
MRS	Multi Robot System
API	Application Program Interface
OpenGl	Open Graphics Library
rab	range and bearing
GUI	Graphic User Interphase
POV-ray	Persistence Of Vision Ray-tracer
CDPS	Cooperative Distributed Problem Solving

ABSTRACT

This report is a study of multi-robot systems. The advantage multi-robot systems pose over single-robot systems both in terms of environmental presence, cost, robustness and redundancy. This report also discusses the importance of coordination in multi-robot system among the robots. Both heterogeneous and homogeneous multi-robot systems are discussed and a homogeneous multi-robot system is also implemented and discussed in detail.

The types of coordination and their incredible importance to improve the performance of multi-robot systems is also discussed. The additional over head of messaging in case of explicit coordination and sensing other robots in addition to environment in case of implicit coordination are meager when compared to improved performance or the ability to perform tasks impossible by single agent systems to do.

An effort has been put to learn more about multi-robot systems by trying to actually program their action from small single-robot systems to heterogeneous multi-agent systems with explicit coordination. ARGoS simulator has been used to evaluate our programs correctness and effectiveness.

Keywords: *Coordination and Cooperation, Multi Agent System, Multi-robot system, Joint Action, Multi Agent Planning, ARGoS simulator,Implicit communication, Explicit communication.*

Contents

1 Introduction 11
2 Related Work 11
3 Multi-Robot Systems 12
3.1 Categories of Multi-robot System 14
3.2 Coordination Strategies for Robots in Multi-Robot System . 14
3.3 Static Coordination 15
3.4 Dynamic Coordination 15
3.4.1 Explicit Coordination 15
3.4.2 Implicit Coordination 16
3.5 Multi-Robot Environment: Cooperative Versus Competitive 16
3.5.1 Cooperation Behaviour 16
3.5.2 Competition Behavior 17
3.6 Communication for Coordination 17
4 Problem Definition 18
5 ARGoS Simulator 18
5.1 Extensibility 20
5.1.1 Modular Architecture of ARGoS 20
5.1.2 Sensors of ARGoS 20
5.1.3 Actuators 20
5.1.4 Additional Modules 21
5.1.5 Simulated Space 22
5.1.6 Entity 22
5.2 Scalability 22
5.2.1 Scatter gather 22
5.2.2 h-dispatch 23
5.3 Running the simulation in ARGoS 23
5.4 Configuration file 23
5.5 Controller file 24
5.6 Loop function 24
6 Implentation of Grid-World Box Pushing 24
6.1 Setting up the simulation 27
6.1.1 Framework 27
6.1.2 Controller 29

6.1.3 Arena 30
6.1.4 Physics Engines 33
6.1.5 Media 33
6.1.6 Visualization 33
6.2 Controller 34
6.2.1 Init 34
6.2.2 Step 35
6.2.3 Search 36
6.2.4 Communication in ARGoS 40
6.2.5 Reaching to specific location 41
6.2.6 Grabbing Blocks 43
6.2.7 Navigate 45
6.2.8 Goal 45
7 Results 46
8 Conclusion 48

List of Figures

1 Types of robot systems 14
2 Coordination strategies in multi-robot systems 15
3 Coordination scenrio in multi-robot systems 16
4 Coordination strategies in multi-robot systems 17
5 Grid world box pushing domain 18
6 The main devices of the foot-bot robot supported by ARGoS [20] . 19
7 The files required to run ARGoS 23
8 The environment for grid-world domain in ARGoS 25
9 The grid-world domain with boxes and robot in ARGoS 25
10 light and heavy boxes 26
11 State diagram for the experiment 28
12 Proximity Sensors 39
13 Range and bearing actuator and sensor functioning 41
14 Finding threshold height for range and bearing sensor and actuator 42
15 Overcoming obstacles in Rab sensor with relay node 42
16 Overcoming obstacles in Rab sensor with relay node 43
17 The grid world domain map for navigation 45
18 Motor Ground Sensor 46
19 Robots removing light weight blocker from the grid block environment 46
20 Team formation to remove heavy block 47
21 Removing Heavy block 48

List of Tables

1 The list of important sensors having by ARGoS 20
2 The list of important actators having by ARGoS 21
3 Message Format 33
4 States of the Robots in course of the experiment 35

1 Introduction

Agent can be thought of as a computer-program, electronic device, electrical device, mechanical device, and even human being which is authorized to perform some specified tasks in place of another. A Multi Agent System (MAS) is a system in which multiple agents that cooperate and coordinate with each other to provide the solution to one or more tasks.

An agent may have so many extra abilities, such as interaction protocol, coordination and cooperation among agents. Agents need to communicate with other agents to maintain the self-reliance among them, and obtain whether a direct or indirect data-exchange. In order to jointly perform a required task or to fulfill a particular goal in multi agent system agent-interaction basically is commanded by different obligation, i.e., cooperation, competition or coexistence. In this work, a robot is considered to realize the agent of the Multi-agent system.

Robots are becoming common occurrence in day today life in many common fields like manufacturing, construction, house maintenance search and rescue, fire fighting, port and warehouse maintenance and also in serious research subjects like planetary exploration etc. Similar to humans when more than one robot as a team work on a complex task by coordinating among each other they may be able to accomplish task impossible for a single robot to accomplish or finish the task in a more efficient manner.

Thus, when a group of robots try to accomplish complex tasks with coordination then the robustness of the solution (due to redundancy) may increase and rate of completion of tasks may also decrease. Such a system of multiple robots working together by coordinating among themselves is a multi-robot system. This coordination in between robots in multi-robot system brings its own set of challenges to effective coordination, such as dynamic events, changing task demands, resource failures, the presence of adversaries, and limited time, energy, computation, communication, sensing, and mobility. Therefore, coordinating a multi-robot team requires overcoming many formidable research challenges.

Furthermore, we simulate multi-robot system using ARGoS (Autonomous Robots Go Swarming) simulator and experience these challenges first hand and attempt to solve some of the problems.

2 Related Work

In this section we have presented the related work. A market-based approach is proposed in [1] for multi-robot task allocation problem for the police force agents. Nonetheless, the authors in [1] have not considered how to deal with heavy blockers. The authors assumed that each task is finished by the single robot. But, in a real circumstance, it isn't the situation and some of the obstacle/road-blockers

(object scattered on road) may be heavy. Thus, we need to guarantee how to clear the road if a heavy obstacle is presented on the road in the rescue environment. In this way managing heavy object becomes essential to deal with. Managing heavy articles is challenging and it requires robot coordination and task allocation to deal with it. The robotic urban search and rescue, i.e., RoboCupRescue simulation environment, type of agents and their responsibilities are discussed in [1–5].

The work presented in [1–5] assumes single task performed by single robot. Task allocation for multi-robot system is an important and challenging problem due to the unpredictable nature of robot environments, sensor failure, robot failure, and dynamically changing task requirements [7–11, 13, 15, 18]. Most of the task allocation solutions propose an auction-based approach wherein robots bid for tasks based on cost functions for performing a task.

Multi-robot coalition formation has been studied by [7], where each robot has some capabilities and a coalition has to perform some task for which some capabilities are required. The authors consider the problem of coalition formation such that there is no coalition imbalance, which happens when a large part of resources is ascribed to any robot in a coalition. A balance coefficient is thus defined and it is used to form a suitable coalition for a given task. In our algorithm, we have assumed that agents have same capabilities to execute a joint action. Thus the notion of imbalance does not arise in our case. Moreover, our algorithm is fully distributed which is not the case with [7].

The work [9] discuss a general task allocation [18] system to intelligently coordinate the group of robots. The main focus of the paper [9] for task allocation is to minimize the three aspects of the system: resource usage, task completion, and communication overhead. In [16] a task allocation algorithm via coalition formation for cooperative distributed problem solving (CDPS) environment is suggested. This algorithm limits the coalition sizes and uses a greedy heuristic to yield a coalition structure that is probably within a bound of the best solution given the limit on the number of agents.

The work [17] discuss a task allocation approach for swarm robotics and simulated the same using ARGoS. The work has considered a transportation task in which the robots transport the specified objects from one place to another. A central auctioneer announces the task identified through surveillance. In our approach, any agent who found the task can be the auctioneer (initiator of coalition formation). In this work we have extended the work dome in [12].

3 Multi-Robot Systems

Multi-robot systems (MRS) in this report consists of multiple mobile robot systems, in which robots work together to accomplish a given task by moving around in the given environment. With the proliferate development of distributed au-

tonomous robot system, in many situations robots are being used to ease the human life, i.e., planetary exploration,manufacturing and construction, medical assistance, search and rescue, and port and warehouse automation. Generally speaking, an MRS can be characterized as a set of robots operating in the same environment. The single as well as multi robot systems have their own advantage and disadvantages. Despite of high capability of individual robots (RHINO, ASIMO, MER-A,BigDog, NAO and PR2 [14], multi robot system have potential advantages over a single robot system, i.e. better special distribution, better overall performance, less time to complete the task, system become more robust with MRS, lower cost of robot in MRS, MRS can exhibit better system reliability, flexibility, scalability and versatility.

A single-robot system has one individual robot that is able to model itself, the environment and their interaction. The robot in a single-robot system is usually designed to deal with a task on its own account. Such robots usually have multiple sensors, which themselves need a complex mechanism and an advanced intelligent control system. Although a single-robot system has a relatively strong performance, some tasks may be inherently too complex or even impossible for it to perform, such as spatially separate tasks. Hence, an inherent restriction to the single-robot system is that it is spatially limited.

A multi-robot system can be either a group of homogeneous(identical) robots or heterogeneous robots that act up on the same environment simultaneously. Some of the advantages of multi-robot system over single-robot system are as follows. A multi-robot system at any given time reads information more about the environment due its presence more places than a single-robot system. Multi-robot system could have better performance than a single-robot system both in time required to complete a task and in energy consumed to do the same. As more robots are introduced robustness and fault-tolerance are introduced and then in a multi-robot system that is not probable in single-robot system. Most of the robots involved in multi-robot systems are simple robots with simple sensors and actuators thus reducing cost of building complex and costly robots involved in single-robot system. As simple robots are involved in multi-robot system many robots can be easily added thus giving scalability.

If robots jointly perform the activity, a team of agents must form before starting the task. Mostly, it is assumed that the entire set of agents is available for task completion. However, these assumptions always may not hold. Hence, an algorithm is needed to form a team of robots.

Multi-agent system and distributed artificial intelligence are two similar fields which are often confused with multi-robot systems. Multi-agent systems usually represent our traditional distributed computer system which are mostly not robots. Distributed artificial intelligence refers to mainly software agents. Robots today are mainly of three types manipulators, mobile robots and humanoid robots.

3.1 Categories of Multi-robot System

Depending on the capabilities and behavior, robots can be categorized into different categories. Homogeneous robots in a multi-agent system represents that all robots involved in the multi-robot system has the same capabilities (same sense of actuators and sensors) not necessarily same physical structures.

Heterogeneous robots in multi-robot system includes robots with different capabilities involved in the set up. Here a particular robot may be specialized for doing a particular part of the task. In general when heterogeneous robots are involved planning becomes more complex. A detail of robot types is depicted in Figure 1

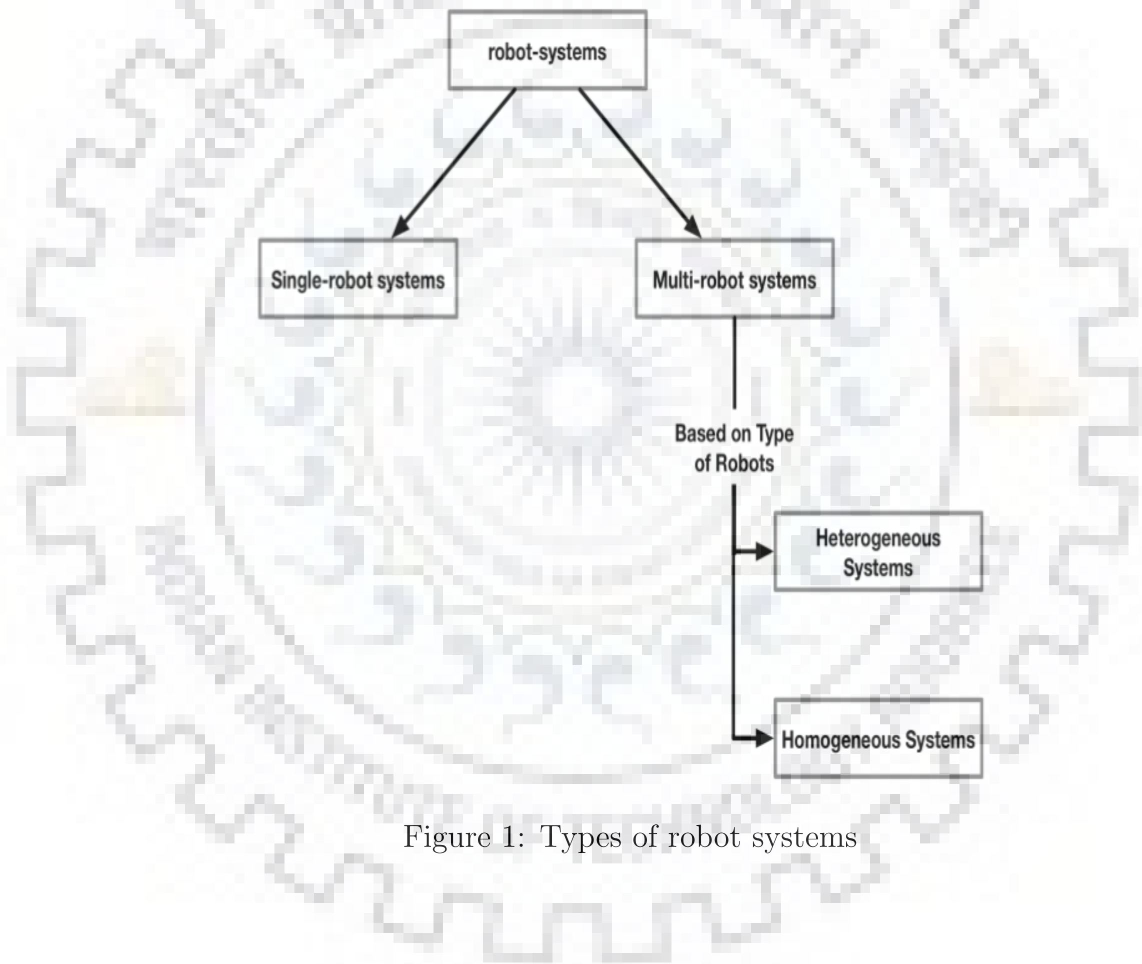


Figure 1: Types of robot systems

3.2 Coordination Strategies for Robots in Multi-Robot System

When there is a bunch of homogeneous or heterogeneous robots in a given environment then in order to take part in a task some form of coordination is necessary between the robots to take advantage of their numbers. To coordinate the robots a simple robust communication with each other. But, communication is over-

head and taxing. So minimalistic communication is required between robots. A classification of coordination strategies is shown in Figure 2

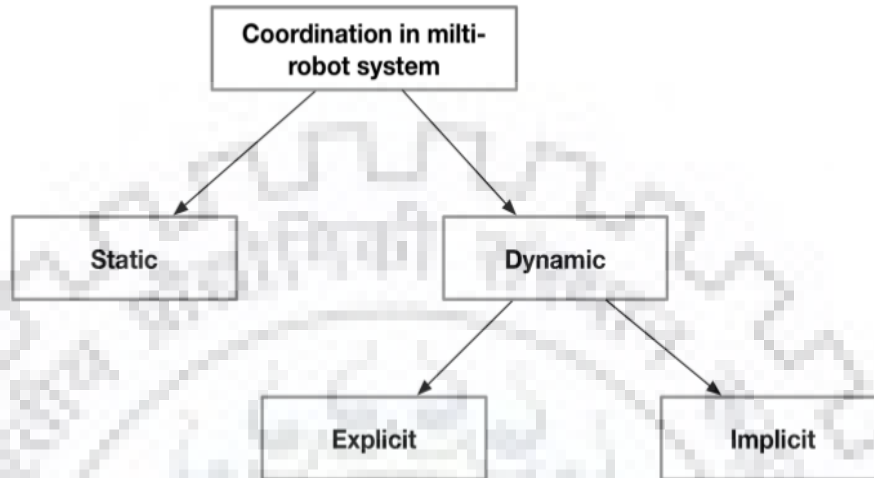


Figure 2: Coordination strategies in multi-robot systems

3.3 Static Coordination

Static coordination also called deliberate coordination or off-line coordination. It usually refers to the rules the a robot in a multi-robot system abides by before even starting the task that helps in better function and coordination in completion of the said task. Say a robot following keep left in path following reduces a lot of path blocking in a path following experiment.

3.4 Dynamic Coordination

Dynamic coordination also called as reactive coordination or on-line coordination is a coordination process that occurs among multi-robot system during the execution of task. Dynamic coordination can be further classified in to explicit and implicit coordination discussed below.

3.4.1 Explicit Coordination

Communication that uses message passing to transfer information or state using uni-cast or broadcast usually using a communication module. Most of the existing coordination are based on explicit coordination.

3.4.2 Implicit Coordination

Implicit coordination is usually associated with implicit communication, which requires the robot to perceive, model and reason other robots' behaviour. Implicit communication refers to the way in which a robot gets its information about other robots in the system. This is usually be achieved by embedding different kinds of sensors in the robot.

3.5 Multi-Robot Environment: Cooperative Versus Competitive

Like human society, there is collective behavior in multi-robot environments. collective behavior is behavior that occurs in response to a common influence or stimulus in relatively spontaneous, unpredictable, unstructured and unstable situations.

The collective behavior includes **cooperative** and **competitive** behavior. In other words, multi-robot environments can be cooperative or competitive.

3.5.1 Cooperation Behaviour

Cooperation refers to a situation whereby multiple robots need to interact together in order to complete a task while increasing the total utility of the system. Alternatively, cooperation is the interaction between the robots, which work towards a common interest or reward. A scenrio is shown in Figure 3.

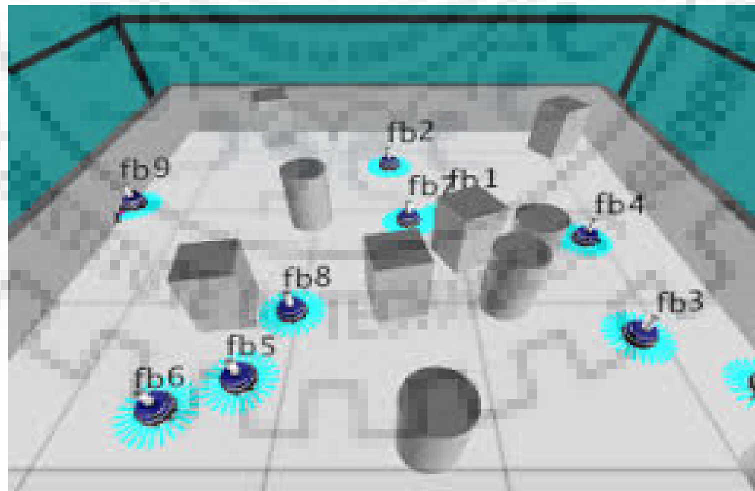


Figure 3: Coordination scenrio in multi-robot systems

The cooperative robots have a joint goal, which gives rise to various sub-goals, eg., multi-robot search and rescue, multi-robot transportation, multi-robot exploration

3.5.2 Competition Behavior

Competition refers to a situation whereby robots compete against each other to best fulfill their own self-interest. Alternatively, robots with conflicting utility functions are in competition with each other. The competitive behavior is the opposite of cooperative behavior. Typical examples of multi-robot competition are two player zero-sum games such as chess, shown in Figure 4.

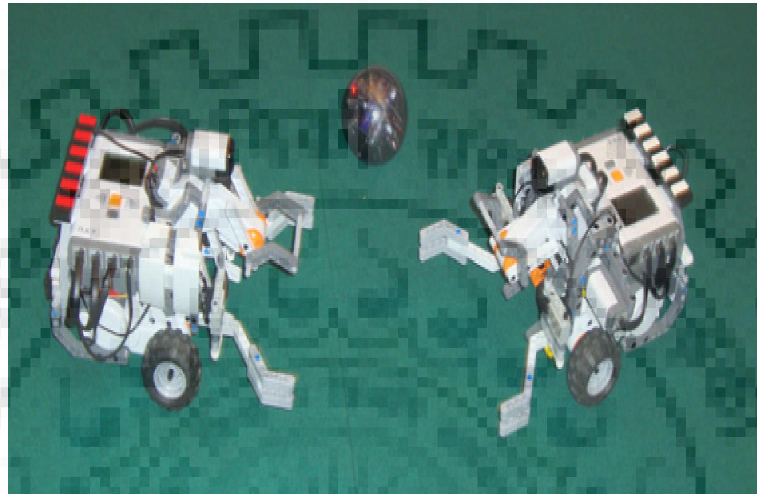


Figure 4: Coordination strategies in multi-robot systems

A multi-robot environment, cooperative or competitive, will need some sort of consensus (a communication mechanism). Robots might be selfish from the sociological point of view, because a single robot tends to make decisions motivated by self-preservation. For instance, consider two robots moving in opposite directions and wanting to cross a narrow passage, but where only one may cross at a time. If the two robots move simultaneously, a congestion or collision will occur. The cooperation can overcome group-think and individual cognitive bias, and this requires some form of coordination. Such coordination can be achieved by communication, which is often used as a rational behavior in multi-robot environments.

3.6 Communication for Coordination

Communication is important for a MRS because it can help robots to be cooperative by learning information that is observed or inferred by others. Explicit communication (another means is implicit communication) must use communication media. However, the communication media cannot always be shared, therefore it is necessary for the robots to obtain exclusive access to them. The problem of communication media sharing is often associated with bandwidth limitation. In Multi-robot system basically, two types of communication is used, i.e, implicit communication and explicit communication. In explicit communication direct

message s are sent among robots while in implicit robots made some change in the environment and via those changes robot receives the signal from the robots.

4 Problem Definition

As a case study to understand the coordination among robots in multi-robot setting, **Box Pushing Domain** is considered, which is studied in paper [6]. The coordination is achieved via direct asynchronous communication, i.,e., explicit. The case study is shown in Figure 5.

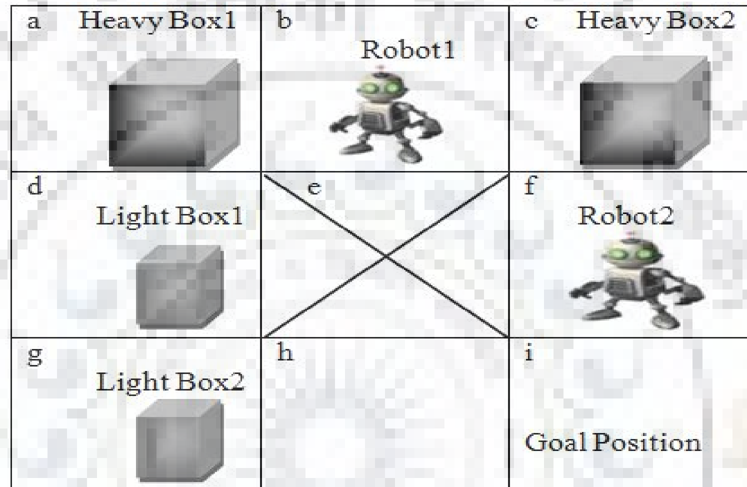


Figure 5: Grid world box pushing domain

In this domain the goal is to move heavy boxes from one location to another by two robot having the same capability. The algorithm for multi-robot coordination is used as proposed by [6]. We implemented the algorithm in ARGoS and apart from the implementation of multi-robot coordination algorithm in a multi-robot simulator, i.e., ARGoS [19], different aspect of the ARGoS's robot are carried out.

5 ARGoS Simulator

Simulation is a cost effective and an efficient way of testing single-robot and multi-robot scenarios. Robotic simulators allow us to create and check the efficiency of algorithm and also causes no loss in case of failure. Many modern simulators include all components required to simulate both robot and environment by using graphic API used for 2d or 3d rendering and precise physics engine to simulate the environment. There are many simulators that are suitable to run a multi-agent simulation like Gazebo/Stage Simulator (Player project), USARSim (Urban Search And Rescue Simulator), WeBots etc . We use a simple simulator called

ARGoS (Autonomous Robots Go Swarming) as it is less computationally demanding and has a modular architecture that allows us to design our robots and being designed as a swarm robot simulator has no problem in creating a large number of robots.

Autonomous Robots Go Swarming(ARGoS) is a multi-robot simulator designed primarily to simulate large scale robotic swarms. Thus it can be used to generate large number of heterogeneous robots. Any multi-robot simulator has two major problems to address namely extensibility (ability to simulate diverse robots) and scalability (ability to simulate numerous robots). By employing a modular architecture and multi-threaded approach to ARGoS solves the above mentioned problems respectively.

The main devices of the foot-bot robot supported by ARGoS are given in Figure 6. One of the most interesting features of the ARGoS simulator is that it allows a user to modify every aspect of a simulation. Because of this flexibility users can select the most suitable modules for the experiment under study. The modular architecture of ARGoS enables addition of new features, such as new robot models, sensors, or actuators, promoting exchange and cooperation among researchers. A unique feature of ARGoS is the possibility to divide the simulated space into non-overlapping sub-spaces, each governed by a separate physics engine. The rules implemented in each physics engine can be customized to optimize the run-time of an experiment. These features make ARGoS suitable for testing the proposed framework.

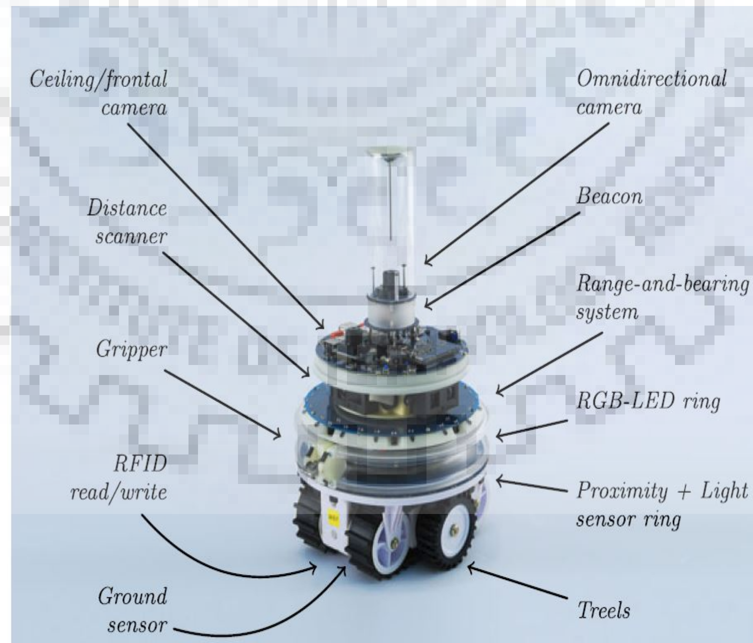


Figure 6: The main devices of the foot-bot robot supported by ARGoS [20]

5.1 Extensibility

The ability to support different types of robots with different sensors and actuators is necessary for a simulator to support heterogeneous multi-robot system. ARGoS Takes care of this by using modular architecture. The modular architecture allows ARGoS to simulate diverse type of robots with sensors and actuators of our choosing.

5.1.1 Modular Architecture of ARGoS

ARGoS is designed as a bunch of modules with essential and optional modules. say in the below picture visualizations is an optional module but other than that all are essential modules. Each module has numerous configurations which work flawless with any configuration in other modules thus giving us the possibility to define any scenario we need. ARGoS uses a set of interfaces and a global simulated space to successfully build and run the simulation.

5.1.2 Sensors of ARGoS

ARGoS consists of the inbuilt multiple sensors present as shown in Table. We can add the sensors we need for every robot we need in the simulation from the available sensors. If we need new sensors we can also design and add them too. The list of important sensors having by ARGoS is shown in Table 1.

Table 1: The list of important sensors having by ARGoS

Sensors	Function
colored blob omnidirectional_camera (rot z only)	A generic omnidirectional camera sensor to detect colored blobs
colored blob perspective camera (default	A generic perspective camera sensor to detect color
differential steering (default)	A generic differential steering sensor
eyebot light (rot z only)	The eye-bot light sensor (optimized for 2D)
eyebot proximity (default)	The eye-bot proximity sensor
footbot base ground (rot z only)	The foot-bot base ground sensor
footbot distance scanner (rot z only)	The foot-bot distance scanner sensor
footbot light (rot z only)	The foot-bot light sensor (optimized for 2D)
footbot motor ground (rot z only)	The foot-bot motor ground sensor
footbot proximity (default)	The foot-bot proximity sensor
footbot turret encoder (default)	The foot-bot turret encoder sensor
ground (rot z only)	A generic ground sensor
light (default)	A generic light sensor
positioning (default)	A generic positioning sensor
proximity (default)	A generic proximity sensor
range and bearing (medium)	The range-and-bearing sensor

5.1.3 Actuators

There are many built in actuators in ARGoS that can be integrated to the available robots and new actuators can be designed and added. The available actuators are

mentioned in the Table 2.

Table 2: The list of important actators having by ARGoS

Actuators	Function
differential steering (default)	The actuator that controls wheel speeds
foot-bot distance scanner (default)	The foot-bot distance scanner actuator
foot-bot gripper (default)	The foot-bot gripper actuator
foot-bot turret (default)	The foot-bot turret actuator
foot-bot turret (default)	The turret actuator
gripper (default)	The gripper actuator
leds (default)	The LEDs actuator
quadrotor position (default)	The quadrotor position actuator
quadrotor speed (default)	The quadrotor actuator
range and bearing (default)	The range and bearing actuator

5.1.4 Additional Modules

Physics Engines

While we decide how the robot acts at different scenarios by detection using the sensors and using its actuators its the physics engines that accounts for the impact the robots have on the environment and the environment have on the robot. A good and accurate physics is very important in any simulator. A multi-physics engine approach is adopted in ARGoS for the development of the simulator. In this approach, multiple engines run in parallel, controlling different aspects/entities of the environment, thus taking advantage of modern multi-core architectures.

ARGoS allows users to run more than one physics engine at the same time with different engines responsible for different region in the simulated space. A robot can be in only one physical engine at a given time but can switch physics engines after a certain task or after a certain time. Simpler physics engines can be used in regions were simpler tasks need to be done and complex engines where complex tasks need to be done.

Media

Media refers to the medium the robots use to communicate with each other. Even noise can be simulated in these communications. The two communication methods used are leds and range and bearing (broadcast messages in line of sight with actuators and sensors)

Visualization

ARGoS uses open source graphic APIs (application programming interface) like OpenGL (Open Graphics Library) and POV-Ray (Persistence of Vision Raytracer) to show the users the simulations.

5.1.5 Simulated Space

That space is where every entity exists. Its also responsible for storing their position, orientation (i.e. physics related information) as well as any other information not related to physics, as for instance visualization information. Simulated space is like a global repository. It has a global view on the en- tire system. Sensors read information from the simulated space according to the permissions they possess. Actuators make changes to the simulated space according to the permissions they possess and physics engines don't allow actuators to make changes that are against the rules of physics. Thus, all data related to the simulation is stored in simulated space.

5.1.6 Entity

Entity refers to robots and obstacles both movable and immovable defined in the simulation. Entities have the following properties. Entities(robots) can belong to only one physics engine at a time. Their interactions are limited to entities on the same physics engine as them. Immovable entities have an exception from the above properties and can belong to multiple engines to ensure consistency of the environment. The entities in ARGoS are box, cylinder, foot-bot, eye-bot, spiri, e-puck etc...

5.2 Scalability

Any simulator that attempts to simulate a swarm of robots must have the capability to support hundreds of robots. Multi-threading allows ARGoS to rise up to the challenge of scalability. Even the implementation of multi- threading in ARGoS is very simple. We needn't mention anything other than the number of threads we want the simulation to use. The simulator core separates threads into master and slave, with the master thread responsible for distributing the tasks to the slave threads.

ARGoS in order to split the task among the number of threads specified uses the following two task splitting algorithms Scatter gather(homogeneous swarm of robots) and h-dispatch(heterogeneous or diverse swarm of robots)

5.2.1 Scatter gather

This is the default algorithm used usually when homogeneous robots are present in the algorithm. As in such scenarios every single robot is going to behave according to similar rules. Each robot is given a thread. Task assignment is simple and is calculated before the beginning of the experiment i.e.. each thread taking care of equal number of robots. There is no need for any dynamic calculation unless robots are added or removed during the run-time of the simulation.

5.2.2 h-dispatch

This method is used when there are diverse robots in the simulation. Here the computational costs will be different for different robots. So, in this method tasks are distributed among the available threads dynamically and each thread gets a new task as soon as it becomes idle. There are two types of thread in this allocation algorithm. A master thread responsible for task allocation across multiple threads (slave threads).

Since the thread allocation is implemented in the core of ARGoS the user does'nt have to change his implementation in order to take advantage of multi-threaded execution but he can mention the which task allocation algorithm to use and the number of threads to be used in the simulation. By default, scatter gather method is used and number threads used by default is one.

5.3 Running the simulation in ARGoS

As a user three files are needed to run an ARGoS simulation out of which two are mandatory. In the Figure 7, the two files, the controller and and the configuration file are mandatory to run the simulation while the loop function is optional.

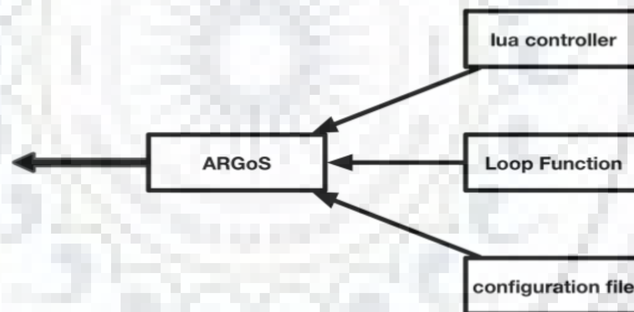


Figure 7: The files required to run ARGoS

5.4 Configuration file

ARGoS uses an *xml* type file with *.argos* extension to define the entire simulation. ARGoS's modular architecture allows us to define size of the environment, movable and immovable obstacles their position, orientation, size weight(in case of movable obstacles). Robots their sensors and actuators and their location is also defined in this file. The duration up to which the experiment has to run, no of threads the experiment will use and the manner in which task allocation happens. Everything is defined in the configuration file.

5.5 Controller file

The robot controller file determines every action of the robots that uses the controller tag that has the said file linked in its param tag. The controller file can either be written in c++ or lua with extension *.cpp* or *.lua* respectively. A controller can be used to control any number of robots but a robot can use only one controller function. Any controller must have the following four functions. An *init* function which is executed every time execute button is pressed and a reset function and destroy function that are executed when reset button is pressed and when a robot is removed from the simulation respectively. The most important function is the step function that is executed to decide the action of the robot it controls at every single step.

5.6 Loop function

The robot controller file determines every action of the robots that uses the controller tag that has the said file linked in its *param* tag. A controller can be used to control any number of robots but a robot can use only one controller. Any controller must have the following four functions. An *init* function which is executed every time execute button is pressed and a reset function and destroy function that are executed when reset button is pressed and when a robot is removed from the simulation respectively. The most important function is the step function that is executed to decide the action of the robot it controls at every single step.

6 Implentation of Grid-World Box Pushing

To implement the case study discussed above, we have to create a grid-world domain in ARGoS first. The first created a grid-world is shown in Figure 8. In this domain centre region which is shown in Gray colour is blocked area and robot cannot enter into this region while roaming in the grid-world. The goal position is black in colour. The goal of the robots is to shift all the boxes at black place (goal position).

The domain initially has 4 heavy and 4 light boxes placed at different location of the grid as shown in Figure 9

The boxes are spread throughout the grid that has to be swept by robots in order to clear the the grid. Depending on the size of the waste barrel or in this case radius of the cylinder one or more robots move the box from the area to be cleaned (white) to the area where waste must be dumped(black). The boxes heavy and light are shown in Figure 10.

Two types of robots are employed to do the above-mentioned task. Spotters that roam around in the white area to search for the waste barrels. Once a spotter has found the waste barrel depending on the size

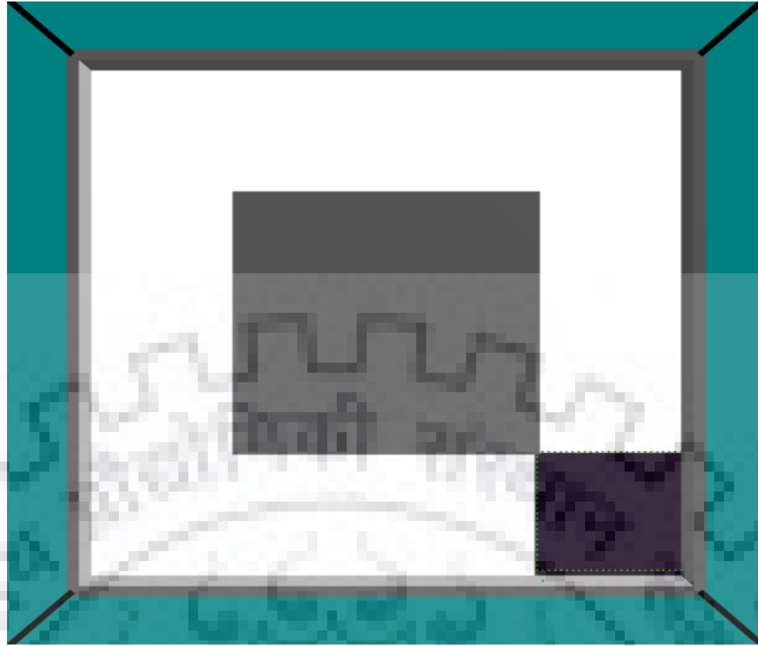


Figure 8: The environment for grid-world domain in ARGoS

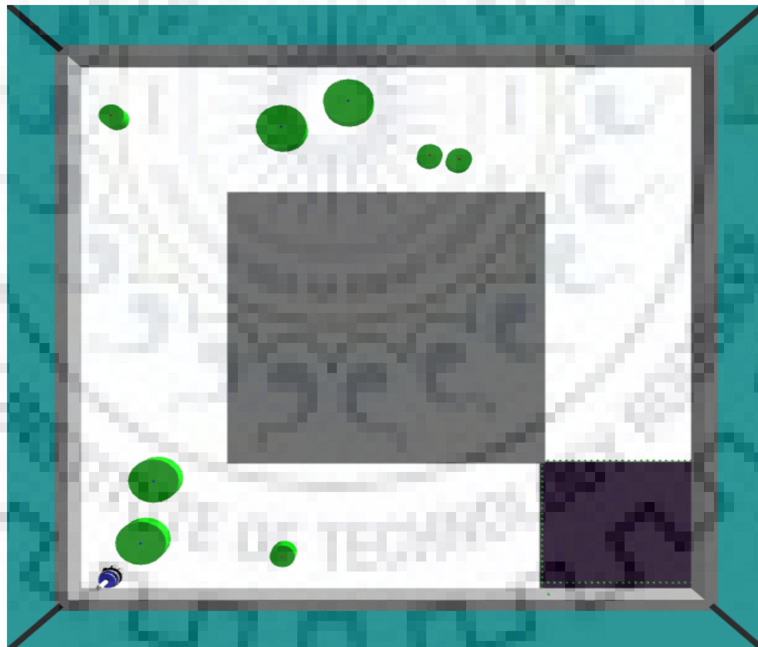


Figure 9: The grid-world domain with boxes and robot in ARGoS

We define the area of the entire arena 10,10,2 with center at 0,0,1. We create the difference between area where waste can be dumped and the area that needed to be cleared of by linking corners.png file to the floor section. Thus, the border region (black region) acts as dump and in addition we define a border with green

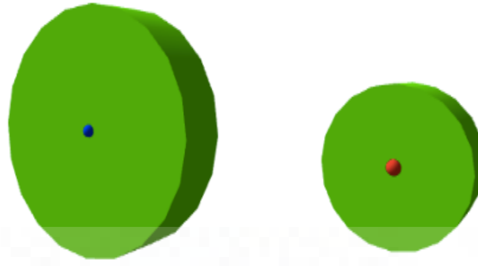


Figure 10: light and heavy boxes

light in a square of side 8.5 separating dump area from the central white area that needs to be cleared. The two types of robots are spotters and helpers are distributed uniformly in the central area that needs to be cleaned and the dump area respectively. The robots have this following sensors and actuators.

Sensors

- Differential steering
- Footbot turret
- Footbot gripper
- leds
- Range and bearing

Actuators

- Positioning
- Colored blob omnidirectional camera
- Footbot motor ground
- Differential steering
- Footbot proximity
- Range and bearing

Now the two sets of robots coordinate and cooperate with each other to clear the area. But to do both we need good communication capabilities. Communication is possible in ARGoS by using its range and bearing sensor and actuator. The bandwidth and range of the sensor can be configured for each robot in the configuration file. We use a three-byte bandwidth message with a range of 15

meters in this experiment for both spotters and helpers. The disadvantage of rab (range and bearing) sensor in ARGoS is the line of sight communication. That is any obstacle which is taller than 0.1 meter either movable/immovable obstacle or a robot will block the communication and every communication is a broadcast.

In order to overcome the two shortcomings we use always obstacles that are of height less than or equal to 0.1 meters and we do not address the problem of robots blocking communication as usually the robots are constantly moving they don't block signals for-ever. Though even this problem can be avoided by using an eye-bot (quadcoptor) to relay all messages thus overcoming any line-of-sight communication problems. We don't use it in this experiment. The second shortcoming being that every transmission a broadcast. We cant develop a router like solution to overcome this problem within the experiment we can develop a solution to address which device the message is meant to be i.e. attaching the receiver and sender address to every packet of information. Every robot in the simulation has an unique string robot.id as an identifier. We use this unique identifier and hash in the interval of 1 to 252.

We follow the state diagram given in Figure 11 to do the intended job.

6.1 Setting up the simulation

In this section the setting up of the grid world box pushing scenario using ARGoS is explained in detail.The experiment is setup using a configuration file with *.argos* extension.

The configuration file is an *xml* syntax code in which all necessary configuration for the simulation, the scenario which we want to simulate and the capabilities and position of initialization of robots is mentioned.

As any *xml* file it is composed of tags necessary and optional tags. The important tags of the configuration file are given below.

1. Framework
2. Controllers
3. Arena
4. Physics Engines
5. Media
6. Visualization

6.1.1 Framework

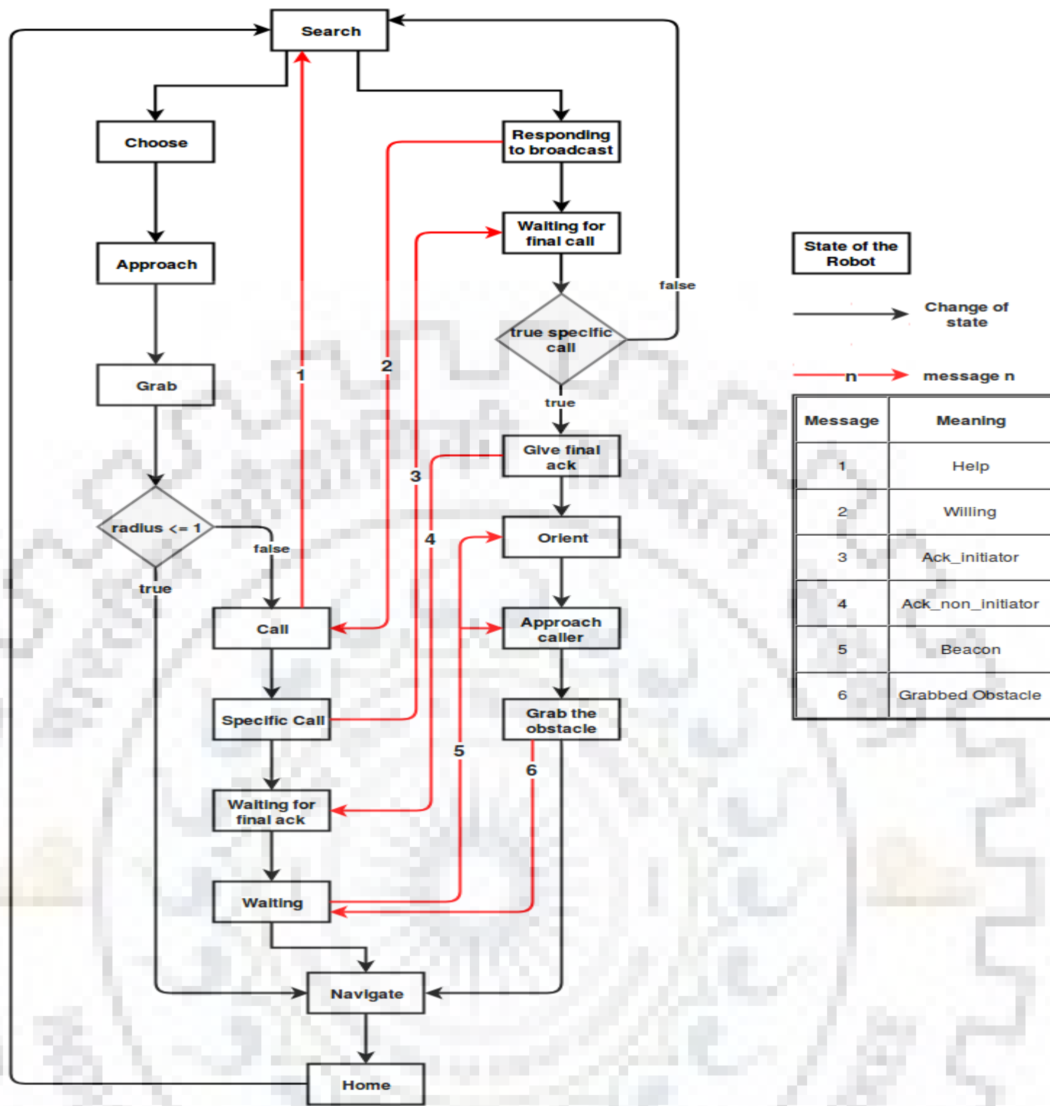


Figure 11: State diagram for the experiment

```
<framework>
<system threads="3" />
<experiment length="0" ticks_per_second="10" random_seed="0" />
</framework>
```

This section is used to fine tune the basic parameters that ARGoS needs to start the simulation like the no of threads, duration of the experiment etc..

The subtag *system* accepts two attributes *threads* indicating number of threads and *method* h-dispatch or scatter the simulation needs to create while running the experiment. By default ARGoS is executed as a single threaded program. But in this experiment as we are using multiple homogeneous robots the *threads* attribute is assigned value equal to the number of robots participating in the

simulation here three. By default ARGoS executes parallelization using scatter gather method which is conducive for homogeneous robot giving us reason to ignore the attribute.

Experiment, an essential *subtag* with attributes such as the duration of the simulation, random seed and number of executions per second. The attribute length refers to number of seconds the experiment will run. Here length tag is zero meaning experiment will keep running till terminated by *ctrl+c*. Ticks per second attribute refers the number of times the simulation step will be executed in a second. Here it is kept at 10. Random seed is the seed value used to create random numbers by ARGoS simulation which are used for random placement of robots and obstacles within mentioned area. If random seed attribute is given a specific value then same random numbers are generated thus helping to run simulation repeatedly thus helping to corroborate results. But here we need to check if the experiment is successful in box pushing at many different initial settings of boxes in the grid. So we keep the random seed value as zero thus using the system time as random seed to create different initialization setting every time we run a program.

6.1.2 Controller

```
<controllers>
<lua_controller id="spot_contr">
<actuators>
<differential_steering implementation="default" />
<footbot_turret implementation="default" />
<footbot_gripper implementation="default" />
<leds implementation="default" medium="leds" />
<range_and_bearing implementation="default" />
</actuators>
<sensors>
<positioning implementation="default" />
<colored_blob_omnidirectional_camera implementation="rot_z_only"
medium="leds" show_rays="true" />
<footbot_motor_ground implementation="rot_z_only" />
<differential_steering implementation="default" />
<footbot_proximity implementation="default" show_rays="false" />
<range_and_bearing implementation="medium" medium="rab"
   show_rays="true" />
</sensors>
<params script="generic.lua" />
</lua_controller>
</controllers>
```

Controller is a very important tag in ARGoS. It determines the number of types of robot that can exist in the scenario. The number of *lua_controller* subtags present in the controller gives us the number of types of robots that can be present in the simulation. Each controller defines the different types of actuators and sensors that can be added to the robot and bind the controller function (.lua file) to each controller via the *param* subtag. These controller function can be similar to different lua_controllers or they can be the same. These controller functions determine every action of every robot added to the controller function throughout the simulation.

This experiment of box pushing is a homogeneous system thus we have only one lua_controller. We add the following sensors to our robot in our simulation to do the below mentioned tasks; an omni directional camera to detect the obstacles, motor ground to read the color of the floor to know if we have reached the goal, differential steering to know the wheel velocity positioning to detect the current location of the robot, proximity sensor to detect obstacles around them and range and bearing sensor to receive messages from other robots.

The following actuators are also needed in the below mentioned manner to control the experiment. A differential steering to control the wheel velocity throughout the experiment, leds to turn on and off the colored leds around the robot, a gripper to grab the obstacles is mounted on a turret that helps to rotate the gripper around the robot and a range and bearing sensor to send messages to other robots.

6.1.3 Arena

```
1  <arena size="5, 5, 2" center="0, 0, 1">
2  <!--Floor design-->
3  <floor id="f" source="image" path="drawing.png" />
4  <!-- central big block-->
5  <box id="bigbox" size="2.5,2.5,0.1" movable="false">
6  <body position="0,0,0" orientation="0,0,0" />
7  <leds medium="leds">
8  <led offset = "-1.25,-1.25,0" anchor = "origin" color = "green" />
9  ...
10 <led offset = "-1.25,-2.5,0" anchor = "origin" color = "green" />
11 </leds>
12 </box>
13
14 <!--Adding borders-->
15 <box id="bn" size="0.1, 5, 0.2" movable="false">
16 <body position="2.5,0,0" orientation="0,0,0" />
17 </box>
18 <box id="bs" size="0.1, 5, 0.2" movable="false">
```

```
19    <body position="-2.5,0,0" orientation="0,0,0" />
20    </box>
21    <box id="be" size="5, 0.1, 0.2" movable="false">
22    <body position="0,-2.5,0" orientation="0,0,0" />
23    </box>
24    <box id="bw" size="5, 0.1, 0.2" movable="false">
25    <body position="0,2.5,0" orientation="0,0,0" />
26    </box>
27
28
29    <!--Robots -->
30    <distribute>
31    <position method="uniform" min="-2.5,-1,0" max="2.5,2.5,0" />
32    <orientation method="uniform" min="0,0,0" max="360,0,0" />
33    <entity quantity="3" max_trials="100">
34    <foot-bot id="spotter-fb" rab_range="15" rab_data_size="3">
35    <controller config="spot_contr" />
36    </foot-bot>
37    </entity>
38    </distribute>
39
40
41    <!-- Big Obstacles -->
42    <distribute>
43    <position method="uniform" min="-2.5,-1,0" max="2.5,2.5,0" />
44    <orientation method="uniform" min="0,0,0" max="0,0,0" />
45    <entity quantity="2" max_trials="100">
46    <cylinder id="cyl1" radius="0.2" height="0.1" movable="true"
        mass="2.5">
47    <leds medium="leds">
48    <led offset=" 0,0,0.1" anchor="origin" color="blue" />
49    </leds>
50    </cylinder>
51    </entity>
52    </distribute>
53
54    <!-- small Obstacles -->
55    <distribute>
56    <position method="uniform" min="-2.5,-1,0" max="2.5,2.5,0" />
57    <orientation method="uniform" min="0,0,0" max="0,0,0" />
58    <entity quantity="4" max_trials="100">
59    <cylinder id="cyl2" radius="0.1" height="0.1" movable="true"
        mass="2.5">
60    <leds medium="leds">
```

```
61    <led offset=" 0,0,0.1" anchor="origin" color="red" />
62    </leds>
63    </cylinder>
64    </entity>
65    </distribute>
66    </arena>
```

Arena is the part where single entity that is going to take part in the experiment is defined and placed. In the size attribute of the arena tag the size of the scenario is mentioned in Cartesian format with scale 1 unit = 1 meter. Thus our scenario becomes a 5 meter square.

The simulation stops abruptly if the robot or obstacles are moved out of this 25 sq. meter defined by the size attribute. To avoid this we use four walls,i.e, unmovable boxes as mentioned in the lines 15, 18, 21 and 24 placed at the edge of the simulation by setting movable tag = 0.

The goal is described by adding a black patch to the floor using an image with black square of side 1.25 meter in *floor* tag as in line 3 of the above code snippet.

A central large square obstacle whose side is 2.5 meter is place in the center of the arena as in line 6. In addition to the black patch which can be detected by the *motor_ground* sensor of the robots, lights are also placed around the goal area to be detected by *colored_blob_omni_directional_camera* sensor of the robot by usin *leds* subtag as in line 7. The medium tag is filled with value of media defined in media tag which is discussed below.

We need to place the robots and blocks randomly around available space except in the goal area. To achieve this we use *distribute* tag. We use the tag distribute both small and large blocks and robots in this simulation in non goal regions as in lines 54, 41 and 29 respectively. Both position and orientation can be distributed in this tag. We use uniform distribution here. Both position and orientation of robots(along z axis) is done. As rotating a cylinder around z axis is ineffective, cylinders are distributed only along position. Small blocks are designed as cylinders with radii 0.1 m while large blocks are designed as cylinder with radii 0.2 m.

The *lua_controller* id from the previous section is bound to the robots which are distributed. *rab_range* and *rab_data_size* are also defined in the foot-bot tag. Rab range determines the range upto which the robot can receive and broadcast message. Here we define it as 15 m. Rab data size denotes the bandwidth of every message that is broadcast by the robot. Here we define the *rab_data_size* is 3 bytes. The messages are of the format given in Table 3

Table 3: Message Format

From Address	To Address	Message
8 bits	8 bits	8 bits

6.1.4 Physics Engines

```
<physics_engines>
<dynamics2d id="dyn2d" />
</physics_engines>
```

Though multiple physics engines can be used. Here we need a simple two dimensional physics engine. As our grid block pushing problem requires only 2d actions, a dynamics2d engine is initialized.

6.1.5 Media

```
<media>
<led id="leds" />
</media>
```

Media is used to implement implicit communications among robots say indicating it is busy or not. While message passing is used for explicit communication media as an implicit communication tool is indispensable.

6.1.6 Visualization

```
<visualization>
<qt-opengl lua_editor ="true">
<camera>
<placement idx="1" position="-0.5,0,0.5" look_at="0,0,0"
   lens_focal_length="20">
<placement idx="2" position="0,0,2" look_at="0,0,0"
   lens_focal_length="20" />
</camera>
</qt-opengl>
</visualization>

</argos-configuration>
```

Visualization is an optional tag. If we want to see the experiment in GUI as graphic then we need visualization. We can avoid using this tab and get out results in terminal too. We use OpenGl to see the graphics.

6.2 Controller

Every controller controls every action of all the robots that use it. Controller can either be a *.lua* or a *.cpp* which was mentioned in the *param* tag in the configuration file. As *generic.lua* was mentioned as the controller file. We design a lua controller to control our robots in this experiment.

Any controller requires four main functions to control any robot throughout the simulation. they are listed below.

- function init(): This function is executed once before starting the simulation. Global values can be initiated in this function.
- function step(): This function is executed in continuous loop throughout the running of the simulation.
- function reset(): This function is run once while the reset button is pressed. This global values initiated in init can be reset to initial values.
- function destroy() :This function is executed once before the simulation is stopped. We can use this function to print results.

We are running a simulation with infinite time. We manually terminate the function using SIGKILL once all blocks have been pushed to goal. Thus, leaving the functions reset and destroy unnecessary. So these two functions are left empty.

6.2.1 Init

```
function init()
self_addr = addr(robot.id)
log(robot.id," = ",id)
state = "search"
prev_state = "dummy"
int_state = "listening"
prev_int_state = "dummy"
robot.colored_blob_omnidirectional_camera.enable()
robot.turret.set_position_control_mode()
end
```

As mentioned earlier *init* is run once before the simulation is started. We need to initialize the some stuff which we need to set-up before start the experiment. ***self_addr*** is an unique address we assign to every robot. This is calculated by converting to unique *robot.id* string to a number between 1 to 252 using the below given formula by the function *addr*. Collision avoidance of the address is taken care by the formula as it is a simple hash formula.

Table 4: States of the Robots in course of the experiment

S.no	State	Internal State
1	Search	Listening
2	Choose	Listening
3	Approach	Listening
4	Grab	Listening
5	Navigate	Listening
6	Call	Listening
7	Specific Call	Listening
8	Waiting for final ack	Listening
9	waiting	Listening
10	Search	Responding to broadcast
11	Search	waiting for final Ack
12	Search	Give final ack
13	Search	Orient
14	Search	Approach caller
15	Search	Grab the obstacle
16	Search	Navigate

$address = \{\sum_{i=1}^{n} ASCII(robot.id[i]) * 2^i\} mod 251 + 1$

We use states to mimic a FSA(Finite State Automata) to control the actions to be taken in various states and the event that causes to change states. We use two dimensional states in this simulation to achieve our goal. All possible states are shown in Table 4. In *init* function we set state as *Search* and Internal State as *Listening*

6.2.2 Step

The *step* function is the most important part of the controller. It calls the required function with respect to the state of the robot. It also prints the change in state of the robot if it occurs. The manner in which the step function works is displayed below.

```
function step()
if state ~= prev_state then
log(self_addr,"=",state)
end
prev_state = state
if state == "search" then
robot.leds.set_all_colors(0,0,0)
search()
elseif state == "choose" then
```

```
robot.leds.set_all_colors("green")
choose()
----other states are also added similiarly
elseif state == "navigate" then
robot.leds.set_all_colors("green")
navigate()
elseif state == "home" then
robot.leds.set_all_colors("green")
home()
end
end
```

6.2.3 Search

The function *search* becomes very important in this simulation as here we control the internal state of the robot in similar manner to step function does the state value. In addition the robot keeps listening for call of help from other robots for pushing heavy blocks and keeps searching for blocks. The functions of the search function are listed below

1. Searching
2. Collision avoidance
3. Waiting for call
4. Changing state

```
1  function search()
2  if int_state ~= prev_int_state then
3  log(self_addr,"=",int_state)
4  end
5  prev_int_state = int_state
6  if int_state == "listening" then
7  sensingLeft =   robot.proximity[3].value +
8  robot.proximity[4].value +
9  robot.proximity[5].value +
10 robot.proximity[6].value +
11 robot.proximity[2].value +
12 robot.proximity[1].value
13
14 sensingRight =   robot.proximity[19].value +
15 robot.proximity[20].value +
```

```
16  robot.proximity[21].value +
17  robot.proximity[22].value +
18  robot.proximity[24].value +
19  robot.proximity[23].value
20  --This ensures that we are not trying to lift already moved boxes--
21  if #robot.colored_blob_omnidirectional_camera >= 1 then
22  check = 0
23  for i = 1, #robot.colored_blob_omnidirectional_camera do
24  if robot.colored_blob_omnidirectional_camera[i].color.green > check
       then
25  check = robot.colored_blob_omnidirectional_camera[i].color.green
26  end
27  end
28  if check == 0 then
29  state = "choose"
30  end
31  end
32  --Obstacle avoidance navigation--
33  if sensingLeft ~= 0 then
34  robot.wheels.set_velocity(7,3)
35  elseif sensingRight ~= 0 then
36  robot.wheels.set_velocity(3,7)
37  else
38  robot.wheels.set_velocity(10,10)
39  end
40
41  --listening part--
42  calls = {}
43  if #robot.range_and_bearing > 0 then
44  for i = 1,#robot.range_and_bearing do
45  if robot.range_and_bearing[i].data[2] == 255 then
46  table.insert(calls, robot.range_and_bearing[i])
47  end
48  end
49  end
50  if #calls > 0 then
51  distance = 10000
52  caller = 0
53  for i = 1, #calls do
54  if calls[i].range < distance then
55  distance = calls[i].range
56  caller = calls[i]
57  end
58  end
```

```
59    end
60    if #calls > 0 and caller ~= 0 then
61    log(self_addr,"received ", msg[caller.data[3]]," from ",
          caller.data[1])
62    int_state = "responding_to_braodcast"
63    end
64
65    elseif int_state == "responding_to_braodcast" then
66    responding_to_braodcast()
67    elseif int_state == "waiting_for_final_call" then
68    waiting_for_final_call()
69    elseif int_state == "give_final_ack" then
70    give_final_ack()
71    elseif int_state == "orient" then
72    orient()
73    elseif int_state == "approach_caller" then
74    approach_caller()
75    elseif int_state == "grab_robot" then
76    grab_robot()
77    elseif int_state == "navigate" then
78    navigate()
79    elseif int_state == "home" then
80    home()
81    end
82    end
```

Searching is done using omnidirectional camera. The blocks are marked by a single red or blue color led light. We read all values of *colored_blob_omni_directional_camera* sensor. If green light is also detected along with red or blue light, it implicitly tells the robot in search state that either the block is in goal state or there is another robot interacting with the block(this is the reason for switching all lights in a robot on when not in search state) These two parts constitute searching from line 20 - 31. A change in state to *choose* represents than a block which needs to be shifted to goal has been found.

Collision Avoidance with other robots or with the central obstacle is important and challenging problem in multi-robot system. In this simulation, proximity sensors of the foot-bot have been used for collision avoidance. The foot-bot has total of 24 number of proximity sensor that help in collecting the data of the object which are nearer to robot as shown in Figure 12. When no obstacle is detected the value of one sensor is zero. When an obstacle is detected then the value of the sensor raises up-to one depending on the closeness of the obstacle. Depending on

the number of the proximity sensor whose value spikes we can know which side the obstacle we are facing. Thus in every step we calculate the value of sensors present in first and fourth quadrant and sum them up as in line 7 and 14. If the values in first quadrant spike then the obstacle is in left side of the robot and the robot turns right else if the obstacle is in the right side of the values in fourth quadrant spike and the robot turns left. If all the proximity sensors read zero then there are no obstacles on the path of the robot and the robot keeps moving straight. This is done between lines 32 -39. The positioning of the proximity sensors on robot are shown in Figure 12.

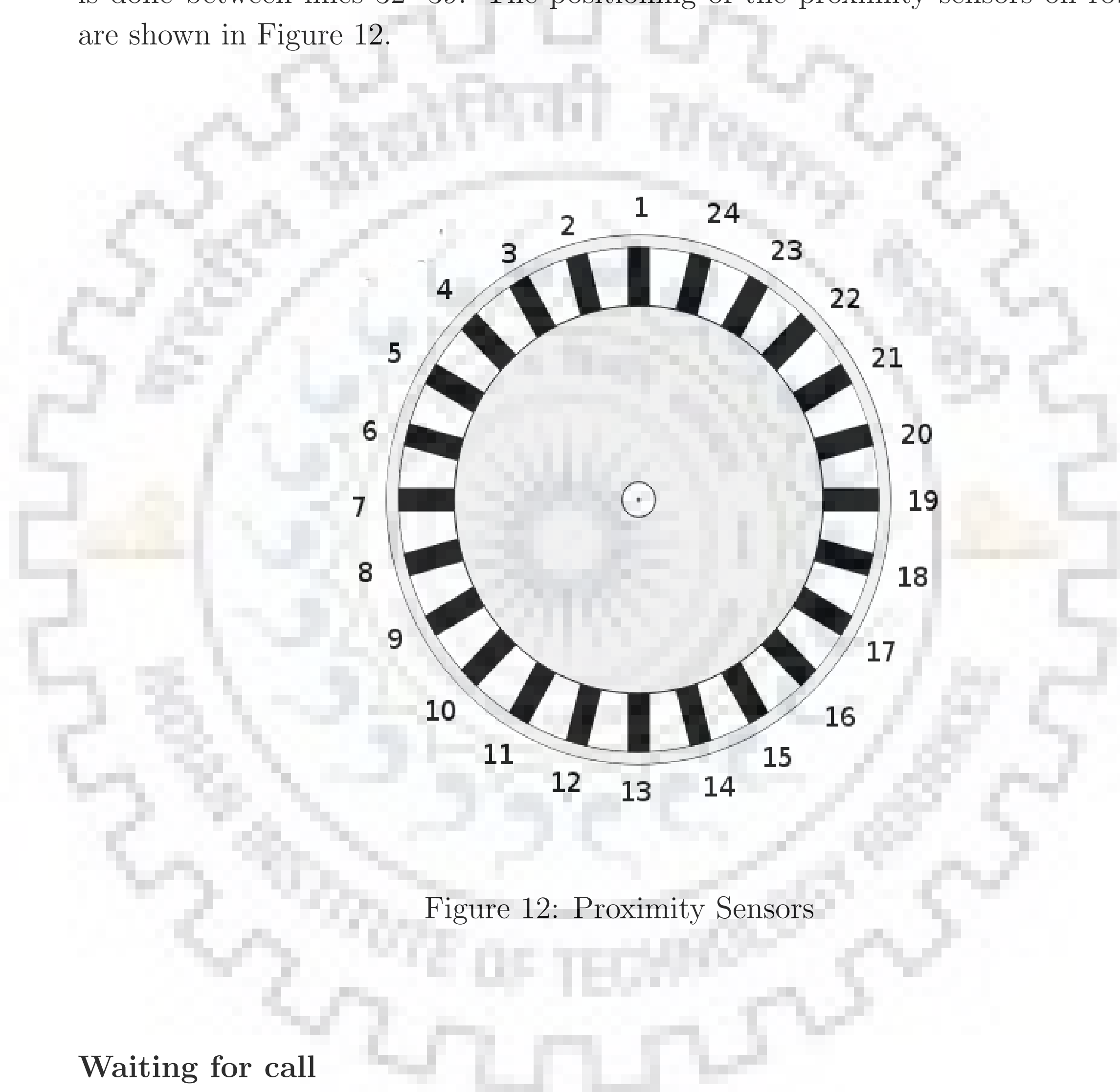


Figure 12: Proximity Sensors

Waiting for call
When the robot is in *search* state and it's internal state remains *listening*. So the robot keeps reading it's RAB sensor in this state for broadcast asking for help.When asked the internal state is switched to *replying_to_broadcast*. This is done in lines 42 -59. In addition to replying to broadcast we also use the *range* part of the message to detect the message that has reached to us from the closest robot and stores that particular message as caller.

Changing State Similiar to *step* function we use the *search* function to call

different function with respect to their internal state as shown in lines 65 -81 To control the speed of robot and to stop at near to some object, we use the formula given in equation 1.

6.2.4 Communication in ARGoS

As shown in the state diagram of our experiment Figure 11. We use in total six types of messages to communicate and coordinate with each other. To compose these messages we use function *compose* which takes as input to address and message. The messages have the following meaning.

Range and bearing sensor and actuator of the foot-bots is used in this experiment to communicate among themselves. Rab actuator allows a robot to broadcast a message of a defined size to all robots within a certain range and in line of sight using the same rab network as the the robot itself. A rab receiver allows a robot to receive messages sent by robots within the same rab network. In addition to the sent message rab sensor allows us to detect the direction and the distance from which the message is being sent.

The *rab* actuator allows only broadcast the address of sender and that of the receiver needs to be specified in every message

```
msg = {}
msg[1] = "Help"
msg[2] = "Willing"
msg[3] = "ACK_initiator"
msg[4] = "ACK_non-initator"
msg[5] = "beacon"
msg[6] = "grabbed road-blocker"
```

We use function compose to To send the message following code is used.
`robot.range_and_bearing.set_data(composed_msg)`

The rab actuator allows a robot to broadcast a message of a defined size to all other robots within a certain range and in line of sight using the same *rab* network as the robot itself. In the experiment, all robots use the same *rab* network and message are of the size 3 bytes (24bits) with a range of 15 meters. A *rab* receiver allows a robot to receive messages sent by robots within the same *rab* network. In addition to the sent message *rab* sensor allows us to detect the direction and the distance from which the message is being sent. As the *rab* actuator allows only broadcast the address of sender and that of the receiver needs to be specified in every message. Every robot in the simulation has a unique string robot.id. This string is used to decide address of every robot using the formula discussed in *init* function.

Thus, address of any robot varies from 1 to 252 and the function is chosen in

such a manner that any collision is avoided in number of robots we use in our experiment (usually 20 to 25). A message format in our experiment contains 8 bit for sending address, 8 bit for receiving address and 8 bit for message content

Figure 13: Range and bearing actuator and sensor functioning

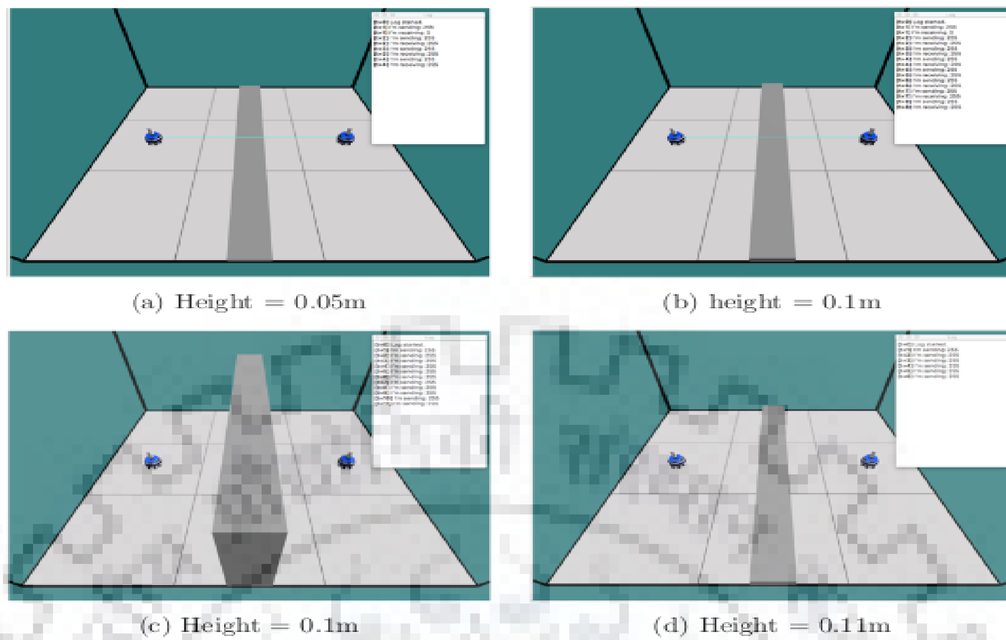
As *rab* is a line of sight communication device. So in order to work with it we need to know its limitations too. At what height an block blocks a line of sight between two robots. So we add obstacles of various height in between the above mentioned two robots and try to find the threshold height till which line of sight is not affected. We find that obstacles upto a height of 0.1 meter will not affect the range and bearing sensor. But if the obstacles are above the height of 0.1 meter then we use the eye bot a quadcoptor robot as a relay accomplish this task. This is shown in Figure 14.

How to make possible communication when line of sight not possible.

6.2.5 Reaching to specific location

Obstacle avoidance can be done using proximity sensors. But reaching for a block after spotting it or reaching for a block after being called to help is a task in itself. Thus we establish two distinct scenarios.

- Reaching Spotted Block
- Reaching to callee Location



(a) Height = 0.05m (b) height = 0.1m

(c) Height = 0.1m (d) Height = 0.11m

Figure 14: Finding threshold height for range and bearing sensor and actuator

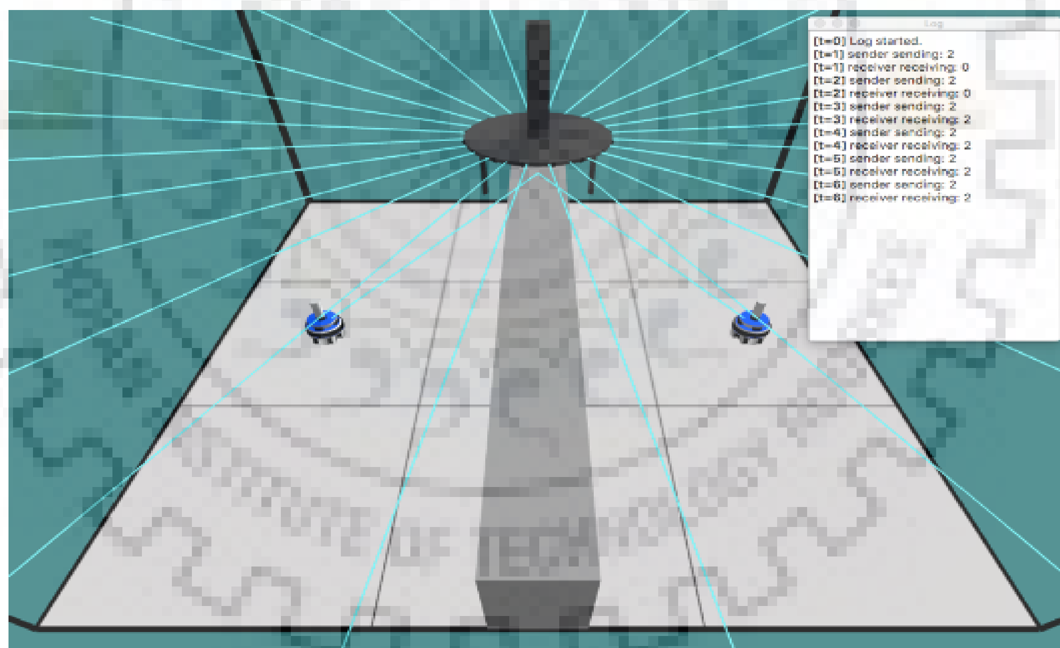


Figure 15: Overcoming obstacles in Rab sensor with relay node

Reaching Spotted Block

When a block is spotted by a robot using *omni_directional_camera*. The sensors also detects the distance and the angle the light is from itself. We first turn the

robot in it's own axis to reduce the angle the light source makes from the front of the robot to a margin less than 20 degrees. Angles measured in radians by the omni_directional_camera sensors as positive in anti-clockwise direction and negative in clockwise direction as shown in Figure 16.

Therefore while traversing toward the block a robot must turn slightly towards right if the angle measured is negative and slightly towards left if the measured angle is positive. And we have to slow down if we near the block.

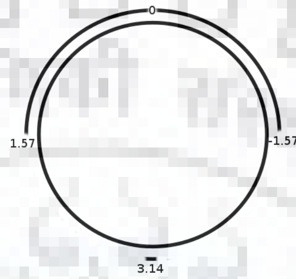


Figure 16: Overcoming obstacles in Rab sensor with relay node

To slow down as we near the obstacle we use the proximity sensor readings according to the below command

$$robot.wheels.set_velocity((1-x)*10,(1-x)*10) \quad (1)$$

where x is the maximum value recorded among the 24 proximity sensors located around the robot.

Reaching to caller Location

When a robot wants to reach and help another while being called to help. The calling robot sets a beacon message while waiting for the called robot to reach it. The robot which want to go and help reads the range and horizontal_bearing of the message. Thus learning the distance and the angle of the message sending robot respectively using its *rab* sensor. Then using similar technique discussed in the previous approach the robot approaches the called robot. But here as the distance between robot and block can far too much obstacle avoidance is given priority over approach. If we face obstacle or any other called block we do avoid it and then continue the approach toward the called robot.

Once we near the robot closer than 50 cm. We use the *omni_directional_camera* sensor to detect the block which has to be moved and approach it as discussed in the previous technique

6.2.6 Grabbing Blocks

While pushing a small block by a single robot or a large block by more than one robot, we have one thing in common i.e we have to grab the blocks. To grab the

block we require two actuators namely *turret* and *gripper*. Gripper is mounted on turret. Objective is to rotate the turret toward the block and activate the gripper.

In order to achieve it the robot uses its proximity sensors and detects the sensor which records the maximum value. As the robot knows the sensor orientations. It turns the turret toward the said orientation and then activates the gripper as described in the code snippet below.

```
function grab()
grip_ang = 200
x = robot.proximity[1]
x.value = 0
pos = 0
for i = 1,24 do
if robot.proximity[i].value >= x.value then
x = robot.proximity[i]
pos = i
end
end
if x.value == 1 then
grip_ang = x.angle
elseif pos >= 1 and pos <= 12 then
robot.wheels.set_velocity(0,0.75)
elseif pos >= 13 and pos <= 24 then
robot.wheels.set_velocity(0.75,0)
end
if grip_ang ~= 200 then

robot.wheels.set_velocity(0,0)
robot.turret.set_rotation(grip_ang)
robot.gripper.lock_negative()
count_time = count_time + 1
end
if count_time == 50 then
robot.gripper.lock_negative()
robot.turret.set_passive_mode()
count_time = 0
if closest.color.blue == 255 then
state = "call"
elseif closest.color.red == 255 then
state = "navigate"
end
end
end
```

6.2.7 Navigate

Once the robot has grabbed the obstacle it uses its positioning sensor to know its location and its orientation with respect to the main axis.

```
x_axis = robot.positioning.position.x
y_axis = robot.positioning.position.y
angle = robot.positioning.orientation.angle
sign = robot.positioning.orientation.axis.z
```

Angle is measure in radians and sign gives if the angle is measured in clockwise or anti-clockwise direction along z axis. A value of +1 represents anti-clockwise measurement. A value of -1 represents clockwise measurement. The robot divides the simulated space into four quadrants as shown in Figure 17. If the robot is present in location two or four it turns toward south and starts to move. If it is present in one or three. It turns toward right and starts moving.

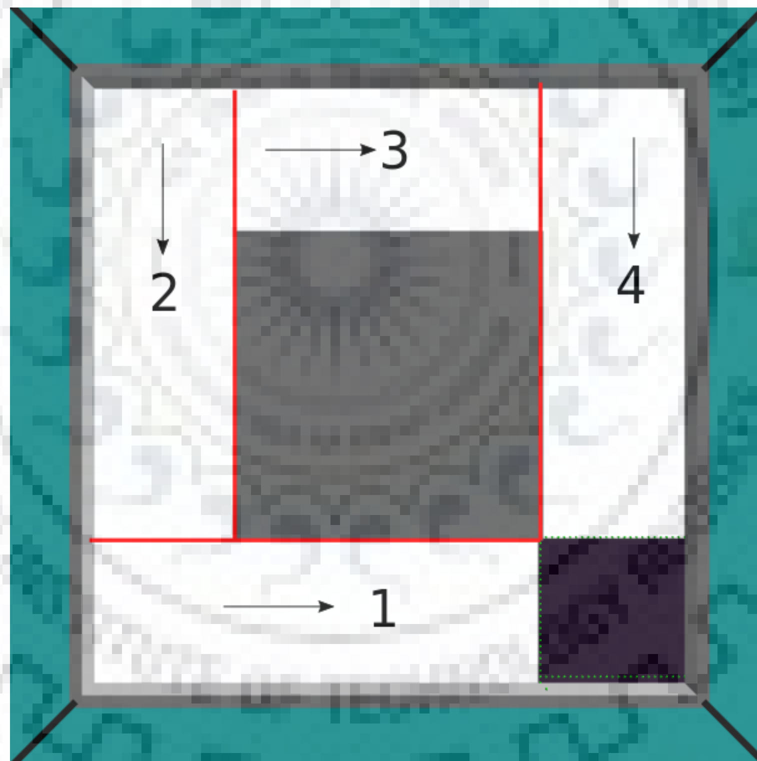


Figure 17: The grid world domain map for navigation

6.2.8 Goal

When it reached the goal state. It detects using its four motor_ground sensors located as shown in the Figure 18. Which read zero whenever the floor is black in

color. Thus detecting the reaching the goal. The robot drops the grabbed block in goal state and change the state to search and continue to search for other blocks.

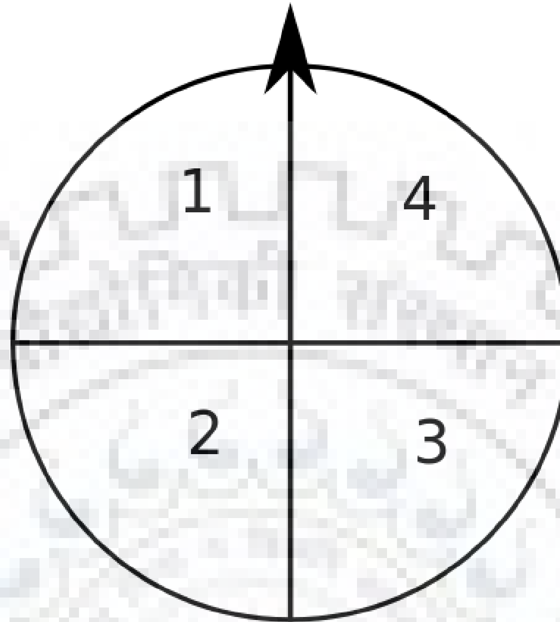
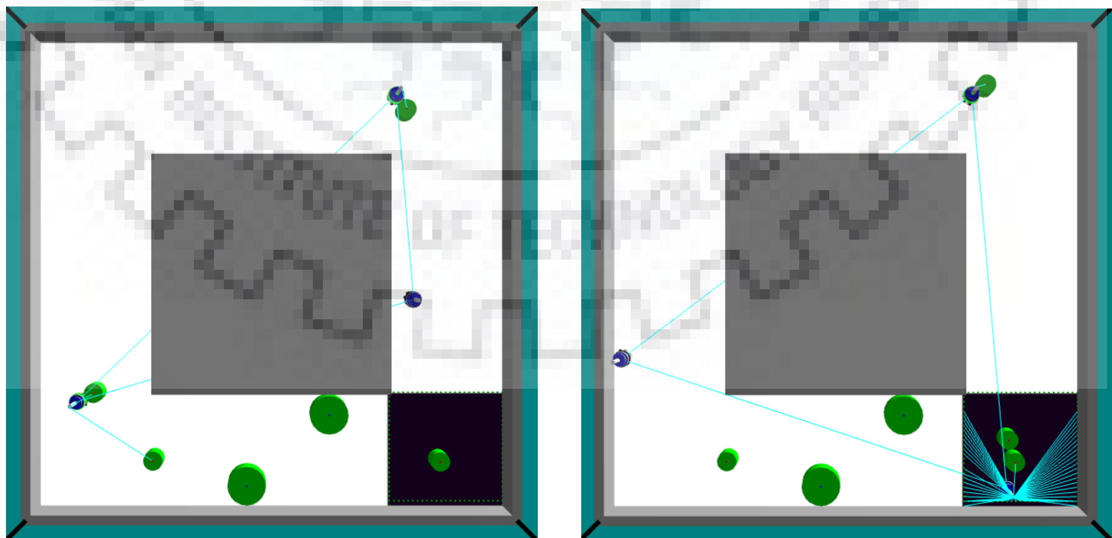


Figure 18: Motor Ground Sensor

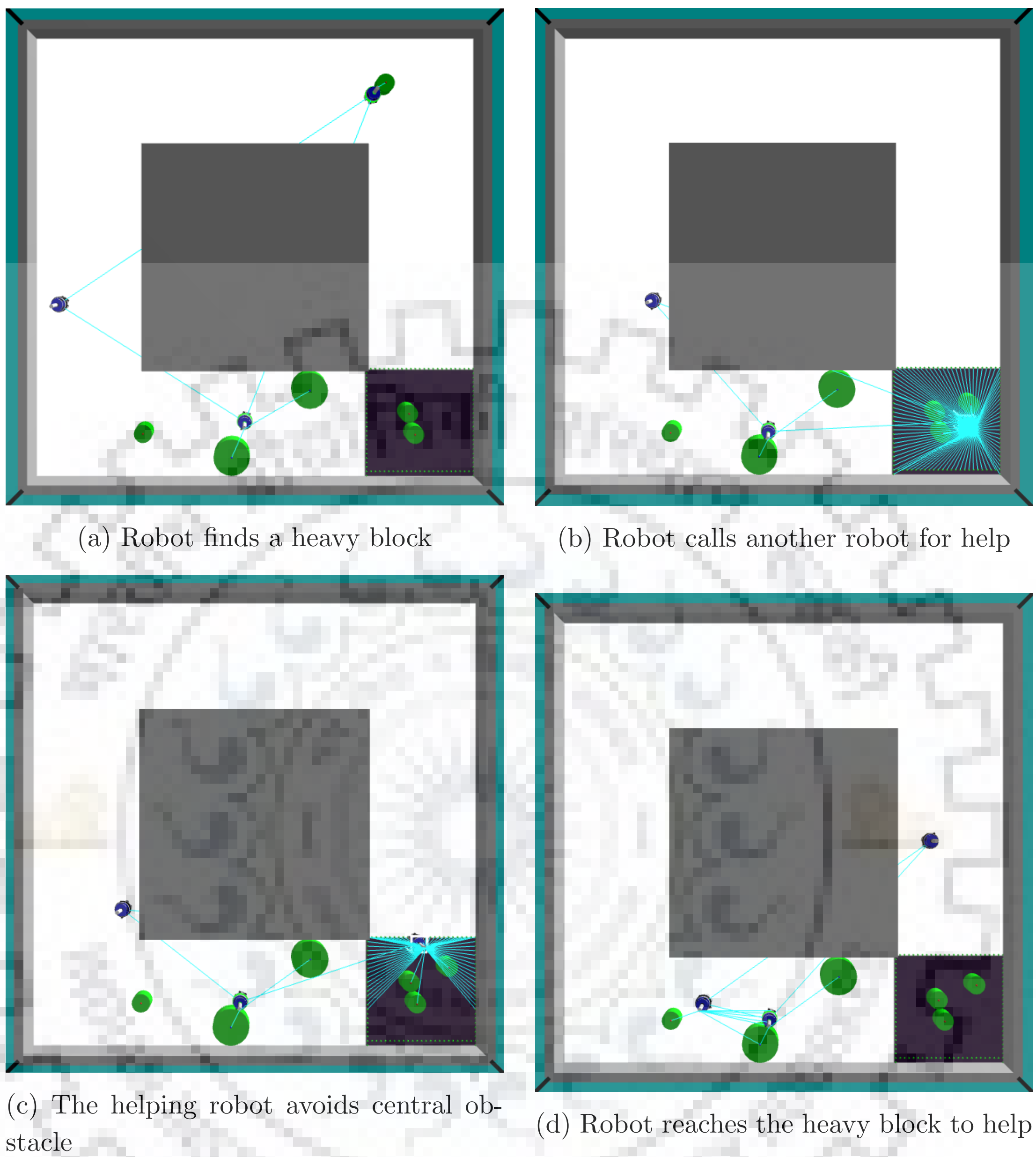
7 Results



(a) Robot finds the light block

(b) Robot drops the block in the goal area

Figure 19: Robots removing light weight blocker from the grid block environment



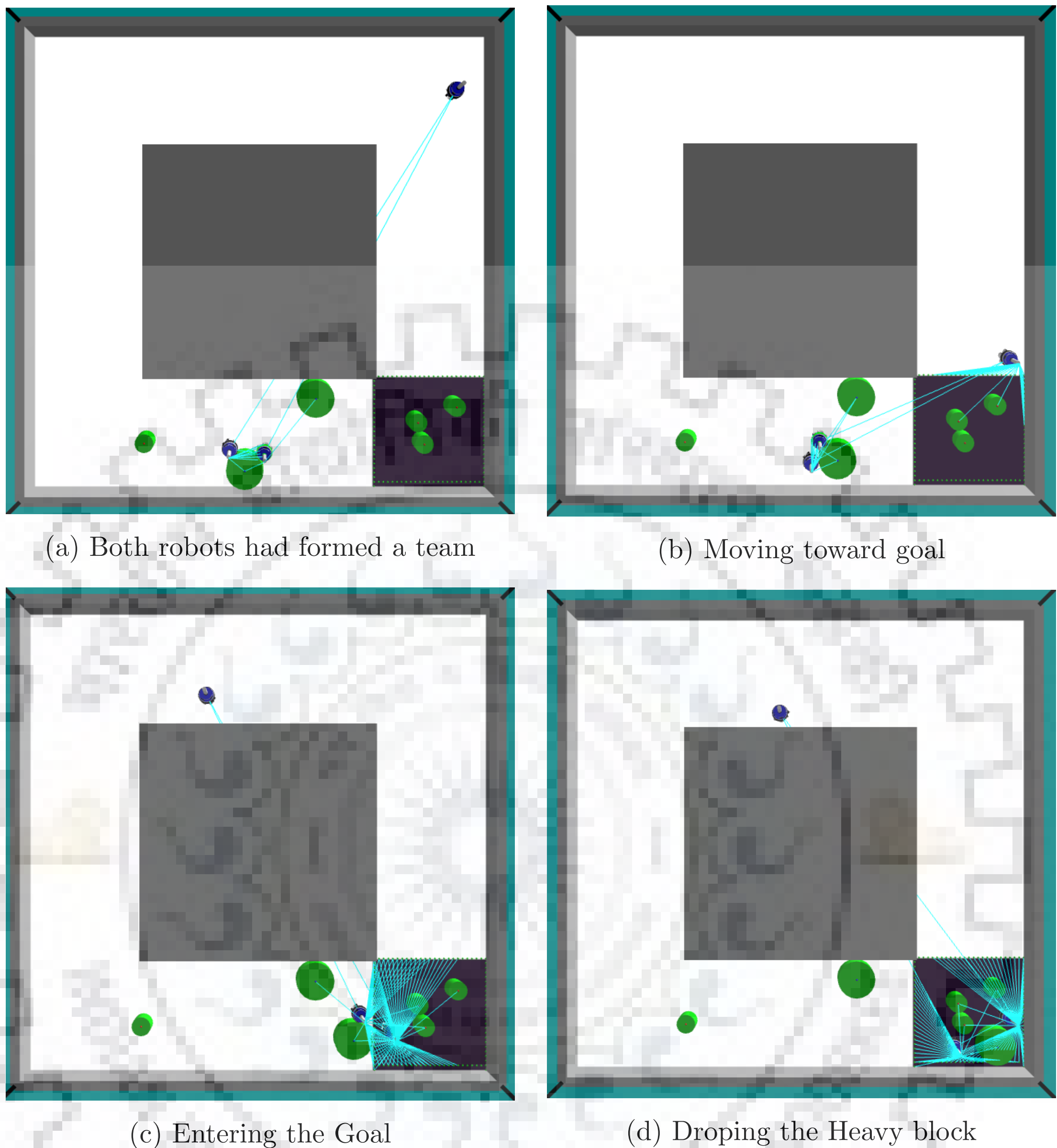
(a) Robot finds a heavy block

(b) Robot calls another robot for help

(c) The helping robot avoids central obstacle

(d) Robot reaches the heavy block to help

Figure 20: Team formation to remove heavy block



(a) Both robots had formed a team

(b) Moving toward goal

(c) Entering the Goal

(d) Droping the Heavy block

Figure 21: Removing Heavy block

8 Conclusion

In this work homogeneous multi-robot system with explicit communication has been explored and discussed. We implemented the distributed algorithm designed for multi-robot coordination in box pushing domain. Challenges like obstacle avoidance and shortest path finding with obstacle avoidance using A* has been implemented. In explicit communication also three-way handshake has been used for team formation for coordinated movement.

Furthermore, we want to extend the work toward mine-sweeping application and room navigation using heterogeneous multi-robot system

References

[1] Sedaghat, M. N., Nejad, L. P., Iravanian, S., & Rafiee, E. (2005, July). Task allocation for the police force agents in robocuprescue simulation. In Robot Soccer World Cup (pp. 656-664). Springer, Berlin, Heidelberg.

[2] Kitano, H., & Tadokoro, S. (2001). Robocup rescue: A grand challenge for multiagent and intelligent systems. AI magazine, 22(1), 39.

[3] Visser, A., Ito, N., & Kleiner, A. (2014, July). Robocup rescue simulation innovation strategy. In Robot Soccer World Cup (pp. 661-672). Springer, Cham.

[4] Nagatani, K., Kiribayashi, S., Okada, Y., Otake, K., Yoshida, K., Tadokoro, S., ... & Kawatsuma, S. (2013). Emergency response to the nuclear accident at the Fukushima Daiichi Nuclear Power Plants using mobile rescue robots. Journal of Field Robotics, 30(1), pp. 44-63.

[5] Liu, Y., & Nejat, G. (2013). Robotic urban search and rescue: A survey from the control perspective. Journal of Intelligent & Robotic Systems, 72(2), pp. 147-165.

[6] Nath, A., & Niyogi, R. (2015, August). An extension of FMAP for joint actions. In Contemporary Computing (IC3), 2015 Eighth International Conference on (pp. 487-492). IEEE.

[7] Vig, L., & Adams, J. A. (2006). Market-based multi-robot coalition formation. In Distributed Autonomous Robotic Systems 7 (pp. 227-236). Springer, Tokyo.

[8] Korsah, G. A., Stentz, A., & Dias, M. B. (2013). A comprehensive taxonomy for multi-robot task allocation. The International Journal of Robotics Research, 32(12), 1495-1512.

[9] Gerkey, B. P., & Matarić, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. The International Journal of Robotics Research, 23(9), 939-954.

[10] Khamis, A., Hussein, A., & Elmogy, A. (2015). Multi-robot task allocation: A review of the state-of-the-art. In Cooperative Robots and Sensor Networks 2015 (pp. 31-51). Springer, Cham.

[11] Lerman, K., Jones, C., Galstyan, A., & Matarić, M. J. (2006). Analysis of dynamic task allocation in multi-robot systems. The International Journal of Robotics Research, 25(3), pp. 225-241.

[12] Nath, A., & Niyogi, R. (2017, September). Design and verification of a collaborative task execution procedure using bpmn modeler. In Advances in Computing, Communications and Informatics (ICACCI), 2017 International Conference on (pp. 103-109). IEEE.

[13] Matarić, M. J., Sukhatme, G. S., & θstergaard, E. H. (2003). Multi-robot task allocation in uncertain environments. Autonomous Robots, 14(2-3), pp. 255-263.

[14] Yan, Z., Jouandeau, N., & Cherif, A. A. (2013). A survey and analysis of multi-robot coordination. International Journal of Advanced Robotic Systems, 10(12), 399.

[15] Tolmidis, A. T., & Petrou, L. (2013). Multi-objective optimization for dynamic task allocation in a multi-robot system. Engineering Applications of Artificial Intelligence, 26(5-6), pp. 1458-1468.

[16] Shehory, O., & Kraus, S. (1998). Methods for task allocation via agent coalition formation. Artificial intelligence, 101(1-2), 165-200.

[17] Shenoy, M. V., & Anupama, K. R. (2017). DTTA-Distributed, Time-division Multiple Access based Task Allocation Framework for Swarm Robots. Defence Science Journal, 67(3), 316.

[18] Das, G. P., McGinnity, T. M., Coleman, S. A., & Behera, L. (2015). A distributed task allocation algorithm for a multi-robot system in healthcare facilities. Journal of Intelligent & Robotic Systems, 80(1), pp. 33-58.

[19] ARGoS:- Multi-robot simulator. `http://www.argos-sim.info/core.php`

[20] Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., ... & Birattari, M. (2012). ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm intelligence, 6(4), pp. 271-295.